REMARKS

Claims 1-19 are pending in the present application. By this Response, claims 1, 6 and 11 are amended. Claims 1, 6 and 11 are amended to correct minor informalities. Reconsideration of the claims in view of the above amendments and the following remarks is respectfully requested.

Amendments are made to the specification to correct errors and to clarify the specification. No new matter is added by any of the amendments to the specification.

I. Examiner Interview

Applicants thank Examiner Yigdall for the courtesies extended Applicants' representatives during the July 20, 2004 telephone interview. During the interview, the differences of the applied references to that of the presently claimed invention were discussed. Examiner Yigdall indicated that the current claims read over the applied reference. Therefore it is Applicants understanding that, pending an update search by Examiner Yigdall, the present claims are now in condition for allowance. The substance of the interview is summarized in the remarks of Section III, which follows.

II. 35 U.S.C. § 112, Second Paragraph

The Office Action rejects claims 6, 7, 11 and 12 under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter, which applicants regard as the invention. This rejection is respectfully traversed. Claims 6 and 11 are amended for clarity by providing proper antecedent basis for the terms identified in the Office Action. Claims 7 and 12 are dependent on claims 6 and 11, therefore the rejection of claims 6, 7, 11 and 12 under 35 U.S.C. § 112, second paragraph has been overcome.

III. 35 U.S.C. § 102, Alleged Anticipation, Claims 1-19

The Office Action rejects claims 1-19 under 35 U.S.C. § 102 as being allegedly anticipated by Darlet et al. (U.S. Patent No. 6,542,167 B1). This rejection is respectfully traversed.

As to claim 1, the Office Action states:

> With respect to claim 1, Darlet discloses a module for use with an application program for use in a data processing system (see the title and abstract), the module comprising:
>
> (a) an export/import list (see column 3, line 64 to column 4, line 7, which shows a list of symbols defining entry points into a module, i.e. an export list, and column 4, lines 39-53, which shows a list of external symbol references, i.e. an import list); and
>
> (b) a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component (see column 6, lines 35-49, which shows a procedure, i.e. a helper function, for resolving unresolved symbols or components during linking, and column 3, lines 10-22, which shows linking may comprise loading).

Office Action dated May 17, 2004, pages 3-4.

Claim 1 reads as follows:

> 1. A module for use with an application program for use in a data processing system, the module comprising:
>
> an export/import list; and
>
> a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. In re Bond, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. In re Lowry, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. Kalman v. Kimberly-Clark Corp., 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). Applicants respectfully submit that Darlet does not identically show each and every claim feature as they are arranged in the claims. Specifically, Darlet does not teach a module for use with an application program for use in a data processing system comprising: an export/import list and a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component.

Darlet is directed to a loading and linking process that allows for the flexible loading and linking of software modules into a memory space, without regard to the order in which symbols are defined and referenced in such software modules, and without generating dangling references. In the system of Darlet, a system is described that includes the steps of locating a symbol reference to a symbol in a first software module and parsing a symbol table to locate an entry for the symbol, the entry for the symbol including a symbol value. Then the system of Darlet locates other symbols in first software module and all other modules, which are loaded and linked to form a single functional group, and creates an entry in the symbol table for each unique symbol. Any symbol that is not unique is updated in the symbol table to link to a previously entered symbol. Once all of the modules are parsed, the symbol table is parsed for any unlinked symbols. If any unresolved symbols are detected, the system scans all of the modules again to determine a link, otherwise the symbols is set to a default, which will cause an error condition when the instruction reference is executed.

Thus, in the system of Darlet, the symbol table exists outside the modules and provides a symbol link to the modules. This teaching of Darlet is also supported by the Office Actions statement that column 3, line 64 to column 4, line 7, which shows a list of symbols defining entry points into a module. Thus, Darlet does not teach a module that is comprised of an export/import list. The Office Action alleges that this feature is taught at column 3, line 64 to column 4, line 7 and column 4, lines 39-53, which read as follows:

> In step 104, symbols defined in the software module are registered in a symbol table, which may be maintained by the linker application (or the operating system) in order to track the use of symbols by software modules. Each software module may define one or more symbols to allow other software modules to access certain locations in the software module (for example, a data structure that may be shared or a library function that may be called). These symbols may be presented as a list of symbol definitions--for example, symbol name and associated value--in the software module for use by the linker.

(Column 3, line 64 to column 4, line 7)

In step 108, symbol references in the software module are resolved. As mentioned earlier, instructions in software modules may contain references to memory locations external to the software module. These external references may be denoted by symbols in the software module that allow a linkage to the external memory location. The linker parses the software module for the use of symbol references in the software module. Where a symbol reference is found in a software module, the symbol reference is resolved by determining the symbol's value from the symbol's entry in the symbol table. Any instructions that use the symbol reference may then be changed based on the symbol value--e.g., the symbol value may be inserted into the instruction. Further details of the symbol resolution procedure will be described below with reference to FIG. 4.

(Column 4, lines 39-53)

In column 3, line 64 to column 4, line 7, Darlet is merely teaching a symbol table, <u>which is maintained by a linker application</u>, that tracks the use of the symbols used by the various software modules. Thus, Darlet teaches a symbol table that is part of the linker application and not a software module which includes an export/import list and a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component. In column 4, lines 39-53, Darlet merely teaches parsing a software module to determine a symbol reference within the module. Then Darlet uses the value in the symbol values in the symbol table to resolve the symbol reference within the software module. Thus, Darlet teaches a linker application that contains a symbol table that is used to resolve symbol references in software modules and not a module that is comprised of an export/import list. A linker application is not a software module which includes an export/import list and a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component.

Furthermore, Darlet does not teach a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component. The Office Action alleges that this feature is taught at column 6, lines 35-49, which reads as follows:

The embodiment according to the present invention also provides a procedure for resolving the unresolved symbol references identified during the exemplary linking process. This procedure is illustrated by the flow chart of FIG. 7, which takes as its starting point the symbol registration step (step 104) from FIG. 1. During symbol registration, the name of the symbol to be registered is compared to the entries already present in the symbol table, to determine whether an entry has already been made for the symbol in the symbol table (step 250). An entry for the symbol will

> already be present in the symbol table where, for example, an unresolved symbol reference exists in an earlier loaded software module (as described above). If there is no entry for the symbol in the symbol table, the symbol is registered as previously described (step 252).

As discussed above, it is the linker application that contains the symbol reference table; thus, any procedure would also be part of the linker application. Additionally, in this section, Darlet is merely describing the process where, during symbol registration, the name of the symbol to be registered is compared to the entries already present in the symbol table, to determine whether an entry has already been made for the symbol in the symbol table. An entry for the symbol will already be present in the symbol table where, for example, an unresolved symbol reference exists in an earlier loaded software module. If there is no entry for the symbol in the symbol table, the symbol is registered, identifying the software module that contains the symbol reference, the address of the instruction using the symbol reference and the base address that should be applied to the address to properly relocate the address. If there already is an entry for the symbol in the symbol table, the definitional information for the symbol from the software module is used to fill in the fields of the entry in the symbol table for the symbol, thus defining the symbol. Thus, the procedure described by Darlet, merely resolves symbols, definition of symbols, and address information that should be applied whenever the symbol is called by a module. Nowhere, in any section of Darlet, is a loader helper function taught as being part of a module.

As to claim 1, the Office Action states:

> With respect to claim 5, Darlet discloses a method for resolving an unresolved module required by a loader in loading an application (see the title and abstract, and column 2, lines 52-65, the method comprising:
>
> (a) determining that a component is unresolvable by the loader (see column 5, lines 36-49, which shows determining that a symbol or component cannot be resolved);
>
> (b) calling a loader helper function to resolve the component (see column 6, lines 35-49, which shows a procedure, i.e. a helper function, for resolving unresolved symbols or components during linking, and column 3, lines 10-22, which shows linking may comprise loading); and
>
> (c) responsive to the loader helper function resolving the component, using the component to resolve a pending import requirement (see column 6, lines 50-67, which shows using the resolved symbol or

component to resolve a pending reference, i.e. a pending import requirement).

Office Action dated May 17, 2004, pages 4-5.

Claim 5, which is representative of the other rejected independent claims 10 and 15, reads as follows:

> 5. A method for resolving an unresolved module required by a loader in loading an application, the method comprising:
> determining that a component is unresolvable by the loader;
> calling a loader helper function to resolve the component; and
> responsive to the loader helper function resolving the component, using the component to resolve a pending import requirement.

Applicants respectfully submit that Darlet does not identically show each and every claim feature as they are arranged in the claims. Specifically, Darlet does not teach a method of resolving an unresolved module required by a loader in loading an application, comprising: determining that a component is unresolvable by the loader, calling a loader helper function to resolve the component; and responsive to the loader helper function resolving the component, using the component to resolve a pending import requirement. As discussed above, Darlet does not teach resolving unresolved modules, but rather resolving symbols in a symbol table that are used by modules. The Office Action alleges that this method is taught in the title, in the abstract and at column 2, lines 52-65, which read as follows:

> System and method for flexible software linking

(Title)

> A loading and linking process allows for the flexible loading and linking of software modules into a memory space, without regard to the order in which symbols are defined and referenced in such software modules, and without generating dangling references. An unloading process allows for software module unloading/unlinking from an already linked set of software modules, also without generating dangling references. A loading/linking system may be used to perform these processes.

(Abstract)

> An exemplary linker and linking process may be implemented as part of an exemplary embodiment according to the present invention. The exemplary linker and linking process allow for the linking of software modules without the need for specific ordering of software modules during

> the linking process ("out-of-order" linking). Software modules may be linked in any order, and symbol references may be resolved without leaving dangling references for undefined symbols. Furthermore, software modules may be "unlinked," such that a software module in a set of previously linked software modules may be replaced by a replacement software module, without sacrificing the integrity of the previously linked software modules or the computing environment.

(Column 2, lines 52-65)

Nowhere in these sections, or any other section of Darlet, is resolving an unresolved module required by a loader in loading an application taught. While Darlet may teach resolving symbol references within a symbol table, without leaving dangling references for undefined symbols, the symbol references are stored in a symbol table and used by all of the modules. Thus, Darlet does not teach resolving an unresolved module required by a loader in loading an application.

Furthermore, Darlet does not teach determining that a component is unresolvable by the loader. As discussed above, Darlet resolves symbol references within a symbol table. Darlet does not determine if a component within a module is unresolvable. Therefore, the calling of the procedure taught by Darlet is to resolve a symbol reference within a symbol table which is part of a linking application and not a software module that determines that a component is unresolvable by the loader and calls a loader helper function to resolve the component. Thus, Darlet does not teach calling a loader helper function to resolve the component of a module.

Thus, Darlet does not teach each and every feature of independent claims 1, 5, 10 and 15 as is required under 35 U.S.C. § 102. At least by virtue of their dependency on independent claims 1, 5, 10 and 15, respectively, Darlet does not teach each and every feature of dependent claims 2-4, 6-9, 11-14 and 16-19. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 1-19 under 35 U.S.C. § 102.

Furthermore, Darlet does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. Absent the Examiner pointing out some teaching or incentive to implement Darlet such that, a module comprises an export/import list and a loader helper function, wherein the loader helper function is callable by a loader to resolve an unresolved component and that an unresolved module is resolved by determining that a component is unresolvable by the loader, calling a loader helper function to resolve the component, and responsive to the loader helper function resolving the component, using the component to resolve a pending import requirement, one of ordinary skill in the art would not be led to modify Darlet to reach the present invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify Darlet in this manner, the presently claimed invention can be reached only through an improper use of hindsight using the Applicants' disclosure as a template to make the necessary changes to reach the claimed invention.

Moreover, in addition to their dependency from independent claims 1, 5, 10 and 15, respectively, the specific features recited in dependent claims 2-4, 6-9, 11-14 and 16-19 are not taught by Darlet. For example, with regard to claims 3, 4, 8, 9, 13, 14, 18 and 19, Darlet does not teach the unresolved component is an unresolved module or an unresolved export. The Office Action alleges that these features are taught at column 6, lines 35-49, and column 3, line 64 to column 4, line 7, shown above. As discussed above, column 6, lines 35-49 merely describes the process where, during symbol registration, the name of the symbol to be registered is compared to the entries already present in the symbol table, to determine whether an entry has already been made for the symbol in the symbol table. In column 3, line 64 to column 4, line 7, Darlet is merely teaching a symbol table, <u>which is maintained by a linker application</u>, that tracks the use of the symbols used by the various software modules. As disclosed by the Office Action, the symbol table shows a list of symbols defining entry points <u>into a module</u>. The resolution of symbols is within the symbol table which is not part of the module, and, thus, Darlet does not teach, a loader helper function is callable by a loader to resolve the unresolved component, where the unresolved component is an unresolved module or an unresolved export.

Additionally, with regard to claims 6, 11 and 16, Darlet does not teach responsive to a determination that the component is unresolvable by the loader helper function, calling a loader helper function in another module to aid in resolving the component. The Office Action alleges that these features are taught at column 6, lines 8-26, and column 5, line 66 to column 6, line 7, which read as follows:

> An example of a reference data structure 310 is shown in FIG. 5. Each symbol in the symbol table 300 may include a reference data structure 310 (even those symbols that are defined, for reasons described

> below). An entry 312 may be made in the reference data structure for each software module which references the symbol and each location in that software module where the symbol reference is used (including appropriate fields for storing addresses for each location).
>
> If the symbol is found in the symbol table, the entry for the symbol in the symbol table is checked to determine if the symbol is pending definition (step 206). If the symbol is still pending, then any instructions using the symbol reference are also changed by inserting the default address into any instructions using the symbol reference (step 211), and an entry is made in the reference data structure (and link status information data structure) for the symbol, identifying the software module and location of the instruction using the symbol reference (step 214). If the symbol is not pending (i.e., the symbol is already defined), the value associated with the symbol is retrieved from the symbol table and used to change those instructions using the symbol reference to the correct value (step 208). An entry may also be made in the reference data structure for the symbol, identifying the software module and location of the symbol reference (step 214), which allows later re-linking of the software module (described below). The linker may then continue to parse the software module for additional symbol references, until all symbol references have been processed (step 216).

(Column 5, line 66 to column 6, line 28)

As discussed above, Darlet teaches that the symbol table and the resolving procedure are part of the linker application and not part of a module. While Darlet may resolve symbol definitions within the symbol table using the reference and definition information within the loaded modules, Darlet does not teach within an unresolved module, calling a loader helper function in another module to aid in resolving the component, responsive to a determination that the component is unresolvable by the loader helper function. Additionally, there is only one resolving procedure taught by Darlet, which is used to resolve undefined symbols within the symbol table. The resolving procedure is part of the linker application and not part of the modules, which the resolving procedure scans to find reference and definition information to populate the field of the symbol table.

Furthermore, with regard to claims 7, 12 and 17, Darlet does not teach responsive to the loader helper function in another module resolving the component, using the component to resolve a pending import requirement. The Office Action alleges that these features are taught at column 6, lines 50-67, which reads as follows:

> If there already is an entry for the symbol in the symbol table, the definitional information for the symbol from the software module (e.g.,

> the symbol's value) is used to fill in the fields of the entry in the symbol table for the symbol (step 254), thus defining the symbol. The linker can then attempt to resolve any symbol references to the symbol that are present in previously loaded software modules, as is indicated in the reference data structure of the symbol table entry (step 256). Since the reference data structure includes the memory location of each instruction using the symbol reference, the symbol value may be used to replace the default address that was previously stored at that memory location. The symbol may also be indicated to no longer be in a "pending" state (step 258). The link status information data structure may be updated based on the symbol definition (step 259), and the process may continue until all symbol definitions have been added to the symbol table (step 260).

In this section, Darlet is merely teaching resolving symbols within the symbol table using the resolving procedure, which are both part of the linker application. Thus, there is only one procedure used to scan the loaded modules for definitional information to populate the symbol table.

Therefore, in addition to being dependent on independent claims 1, 5, 10 and 15, respectively, dependent claims 2-4, 6-9, 11-14 and 16-19 are also distinguishable over Darlet by virtue of the specific features recited in these claims. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 2-4, 6-9, 11-14 and 16-19 under 35 U.S.C. § 102.

IV. Conclusion

It is respectfully urged that the subject application is patentable over the prior art of record and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

Respectfully submitted,

DATE: July 29, 2004

Francis Lammes

Francis Lammes
Reg. No. 55,353
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 367-2001
Agent for Applicants